intel

APPLICATION NOTE

AP-166

October 1983

USING THE 8291A GPIB TALKER/LISTENER

ORDER NUMBER: 230832-001

INTRODUCTION

This application note explains the Intel® 8291A GPIB (General Purpose Interface Bus) Talker/Listener as a component, and shows its use in GPIB interface design tasks.

The first section of this note presents an overview of IEEE 488 (GPIB). The second section introduces the Intel® GPIB component family. A detailed explanation of the 8291A follows. Finally, some application examples using the component family are presented.

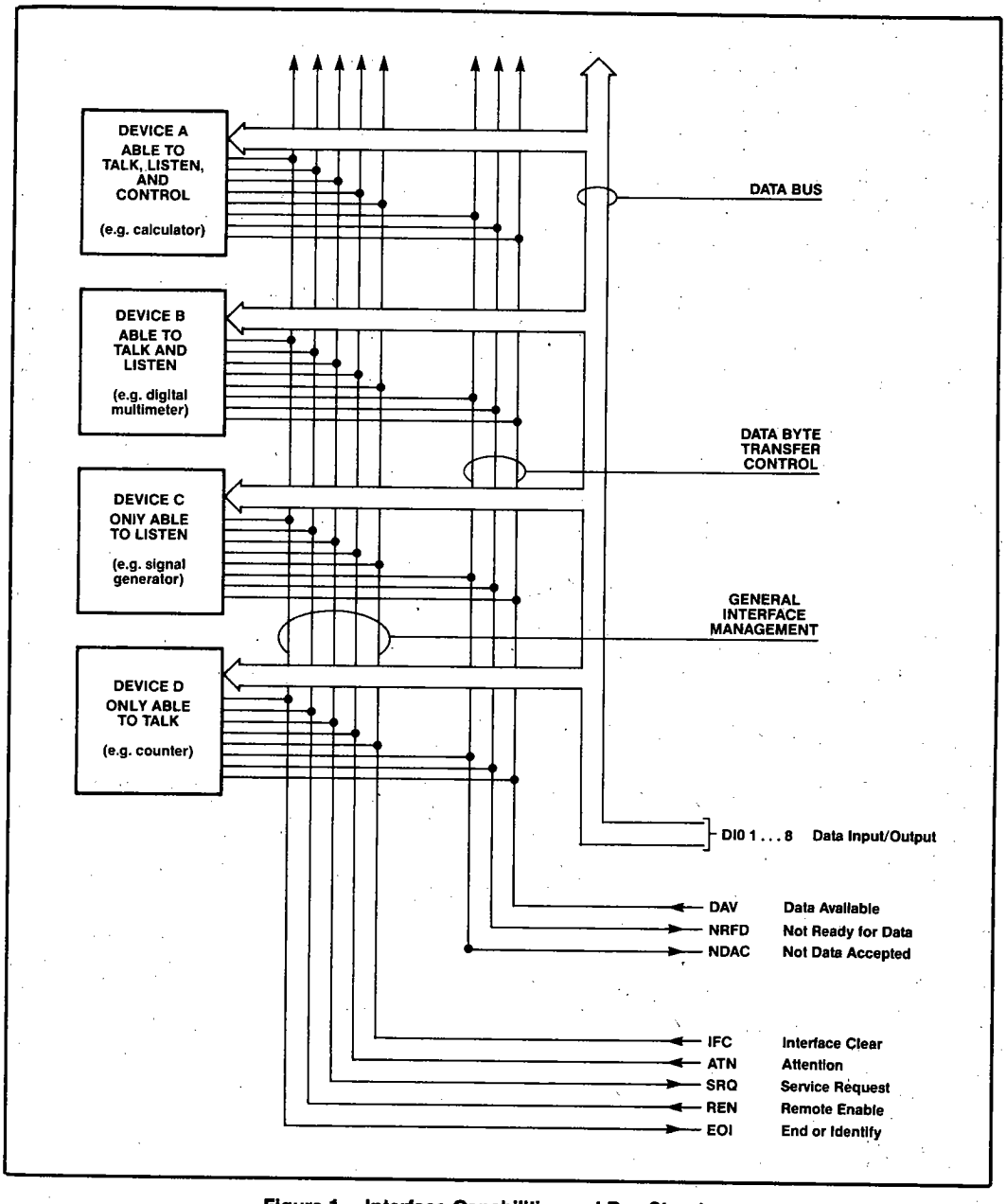


Figure 1. Interface Capabilities and Bus Structure

OVERVIEW OF IEEE 488/GPIB

The GPIB is a parallel interface bus with an asynchronous interlocking data exchange handshake mechanism. It is designed to provide a common communication interface among devices over a maximium distance of 20 meters at a maximum speed of 1 Mbps. Up to 15 devices may be connected together. The asynchronous interlocking handshake dispenses with a common synchronization clock, and allows intercommunication among devices capable of running at different speeds. During any transaction, the data transfer occurs at the speed of the slowest device involved.

The GPIB finds use in a diversity of applications requiring communication among digital devices over short distances. Common examples are: programmable instrumentation systems, computer to peripherals, etc.

The interface is completely defined in the IEEE Std.-488-1978.

A typical implementation consists of logical devices which talk (talker), listen (listeners), and control GPIB activity (controllers).

Interface Functions

The interface between any device and the bus may have a combination of several different capabilities (called 'functions'). Among a total of ten functions defined, the Talker, Listener, Source Handshake, Acceptor Handshake and Controller are the more common examples. The Talker function allows a device to transmit data. The Listener function allows reception. The Source and Acceptor Handshakes, synchronized with the Talker and Listener functions respectively, exchange the handshake signals that coordinate data transfer. The Controller function allows a device to activate the interface functions of the various devices through commands. Other interface functions are: Service request, Remote local, Parallel poll, Device clear and Device trigger. Each interface may not contain all these functions. Further, most of these functions may be implemented to various levels (called 'subsets') of capability. Thus, the overall capability of an interface may be tailored to the needs of the communicating device.

Electrical Signal Lines

As shown in Figure 1, the GPIB is composed of eight data lines (D08-D01), five interface management lines (IFC, ATN, SRQ, REN, EOI), and three transfer control lines (DAV, NRFD, NDAC).

The eight data lines are used to transfer data and commands from one device to another with the help of the management and control lines. Each of the five interface management lines has a specific function.

ATN (attention) is used by the Controller to indicate that it (the controller) has access to the GPIB and that its output on the data lines is to be interpreted as a command. ATN is also used by the controller along with EOI to indicate a parallel poll.

SRQ (service request) is used by a device to request service from the controller.

REN (remote enable) is used by the controller to specify the command source of a device. A device can be issued commands either locally through its front panel or by the controller.

EOI (end or identify) may be used by the controller as well as a talker. A controller uses EOI along with ATN to demand a parallel poll. Used by a talker, EOI indicates the last byte of a data block.

IFC (interface clear) forces a complete GPIB interface to the idle state. This could be considered the GPIB's "interface reset." GPIB architecture allows for more than one controller to be connected to the bus simultaneously. Only one of these controllers may be in command at any given time. This device is known as the controller-in-charge. Control can be passed from one controller to another. Only one among all the controllers present on a bus can be the system controller. The system controller is the only device allowed to drive IFC.

Transfer Control Lines

The transfer control lines conduct the asynchronous interlocking three-wire handshake.

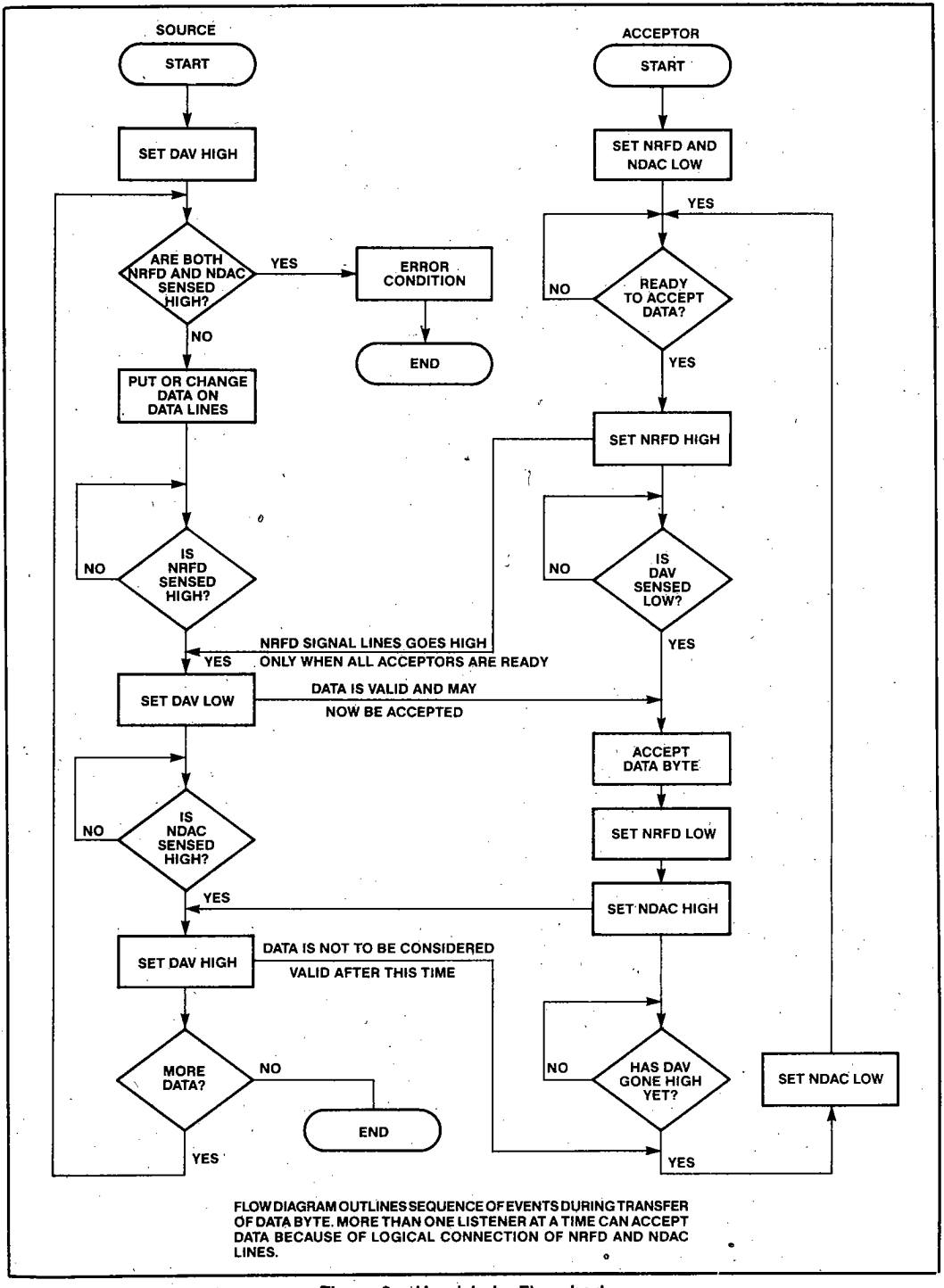
DAV (data valid) is driven by a talker and indicates that valid data is on the bus.

NRFD (not ready for data) is driven by the listeners and indicates that not all listeners are ready for more data.

NDAC (not data accepted) is used by the listeners to indicate that not all listeners have read the GPIB data lines yet.

The asynchronous 3-wire handshake flowchart is shown in Figure 2. This is a concept fundamental to the asynchronous nature of the GPIB and is reviewed in the following paragraphs.

Assume that a talker is ready to start a data transfer. At the beginning of the handshake, NRFD is false indicating that the listener(s) is ready for data. NDAC is true indicating that the listener(s) has not accepted the data, since no data has been sent yet. The talker places data on the data lines, waits for the required settling time, and then indicates valid data by driving DAV true. All active listeners drive NRFD true indicating that they are not



FLOW DIAGRAM OUTLINES SEQUENCE OF EVENTS DURING TRANSFER OF DATA BYTE. MORE THAN ONE LISTENER AT A TIME CAN ACCEPT DATA BECAUSE OF LOGICAL CONNECTION OF NRFD AND NDAC LINES.

Figure 2. Handshake Flowchart

230832-001

ready for more data. They then read the data and drive NDAC false to indicate acceptance. The talker responds by deasserting DAV and readies itself to transfer the next byte. The listeners respond to DAV false by driving NDAC true. The talker can now drive the data lines with a new data byte and wait for NRFD to be false to start the next handshake cycle.

Bus Commands

When ATN and DAV are true data patterns which have been placed by the controller on the GPIB, they are interpreted as commands by the other devices on the interface. The GPIB standard contains a repertory of commands such as MTA (My Talk Address), MSA (My Secondary Address), SPE (Serial Poll Enable), etc. All other patterns in conjunction with ATN and DAV are classified as undefined commands and their meaning is user-dependent.

Addressing Techniques

To allow the controller to issue commands selectively to specific devices, three types of addressing exist on the GPIB: talk only/listen only (ton/lon), primary, and secondary.

Ton/lon is a method where the ability of the GPIB interface to talk or listen is determined by the device and not by the GPIB controller. With this method, fixed roles can be easily designated in simple systems where reassignment is not necessary. This is appropriate and convenient for certain applications. For example, a logic analyzer might be interfaced via the GPIB to a line printer in order to document some type of failure. In this case, the line printer simply listens to the logic analyzer, which is a talker.

The controller addresses devices through three commands, MTA (my talk address), MLA (my listen address), and MSA (my secondary address). The device address is imbedded in the command bit pattern. The device whose address matches the imbedded pattern is enabled. Some devices may have the same logical talk and listen addresses. This is allowable since the talker and listener are separate functions. However, two of the same functions cannot have the same address.

In primary addressing, a device is enabled to talk (listen) by receiving the MTA (MLA) message.

Secondary addressing extends the address field from 5 to 10 bits by allowing an additional byte. This additional byte is passed via the MSA message. Secondary addressing can also be used to logically divide devices into various subgroups. The MSA message applies only to the device(s) whose primary address immediately preceed it.

INTEL'S® GPIB COMPONENTS

The logic designer implementing a GPIB interface has, in the past, been faced with a difficult and complex discrete logic design. Advances in LSI technology have produced sophisticated microprocessor and peripheral devices which combine to reduce this once complex interface task to a system consisting of a small set of integrated circuits and some software drivers. A microprocessor hardware/software solution and a high-level language source code provide an additional benefit in end-product maintenance. Product changes are a simple matter of revising the product software. Field changes are as easy as exchanging EPROMS.

Intel® has provided an LSI solution to GPIB interfacing with a talker/listener device (8291A), a controller device (8292), and a transceiver (8293). An interface with all capabilities except for the controller function can be built with an 8291A and a pair of 8293's. The addition of the 8292 produces a complete interface. Since most devices in a GPIB system will not have the controller function capability, this modular approach provides the least cost to the majority of interface designs.

Overview of the 8291A GPIB Talker/Listener

The Intel® 8291A GPIB Talker/Listener operates over a clock range of 1 to 8 MHz and is compatible with the MCS-85, iAPX-86, and 8051 families of microprocessors.

A detailed description of the 8291A is given in the data sheet.

The 8291A implements the following functions: Source Handshake (SH), Acceptor Handshake (AH), Talker Extended (TE), Service Request (SRQ), Listener Extended (LE), Remote/Local (RL), Parallel Poll (PP2), Device Clear (DC), and Device Trigger (DT).

Current states of the 8291A can be determined by examining the device's status read registers. In addition, the 8291A contains 8 write registers. These registers are shown in Figure 3. The three register select pins RS3-RS0 are used to select the desired register.

The data — in register moves data from the GPIB to the microprocessor or to memory when the 8291A is addressed to listen. When the 8291A is addressed to talk, it uses the data - out register to move data onto the GPIB. The serial poll mode and status registers are used to request service and program the serial poll status byte.

A detailed description of each of the registers, along with state diagrams can be found in the 8291A data sheet.

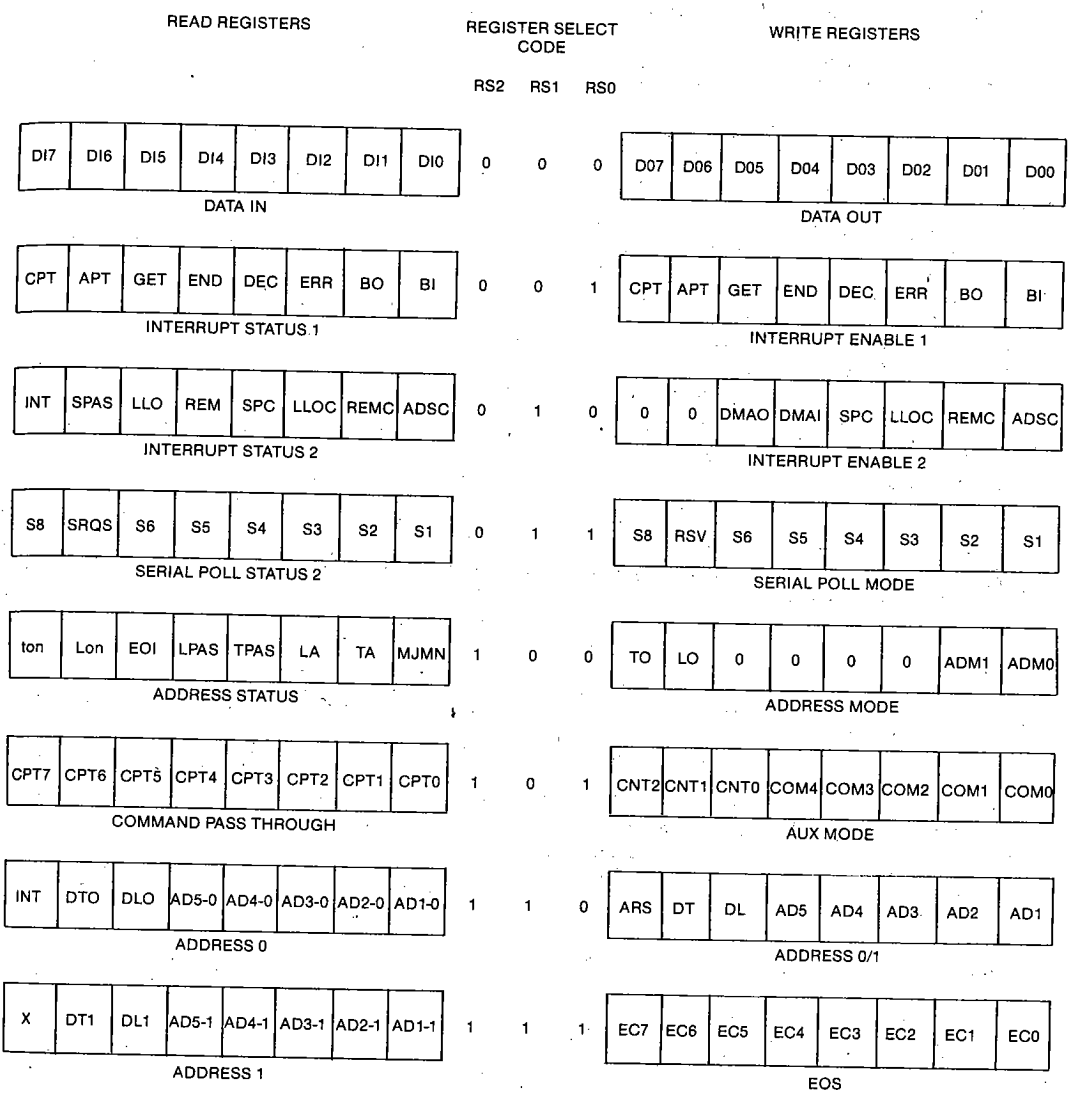


Figure 3. 8291A REGISTERS

Address Mode

The address mode and status registers are used to program the addressing modes and track addressing states. The auxiliary mode register is used to select a variety of functions. The command pass through register is used for undefined commands and extended addresses. The address 0/1 register is used to program the addresses to which the 8291A will respond. The address 0 and address 1 registers allow reading of these programmed addresses plus trading of the interrupt bit. The EOS register is used to program the end of sequence character.

Detailed descriptions of the addressing modes available with the 8291A are described in the 8291A data sheet. Examples of how to program these modes are shown below.

1. MODE: — Talker has single address of 01H
   — Listener has single address of 02H

CPU WRITES TO:	PATTERN	COMMENT
Address Mode Register	0000 0001	Select Mode 1 Addressing
Address 0/1 Register	0010 0001	Major is Talking. Address = 01H
Address 0/1 Register	1100 0010	Minor is Listener. Address = 02H

2. MODE: — Talker has single address of 01H
   — Listener has single address of 02H

CPU WRITES TO:	PATTERN	COMMENT
Address Mode Register	0000 0001	Select Mode 1 Addressing
Address 0/1 Register	0100 0010	Major is Listener. Address = 02H
Address 0/1 Register	1010 0001	Minor is Talking. Address = 01H

Note that in both of the above examples, the listener will respond to a MLA message with five least significant bits equal to 02H and the talker to a 01H.

3. MODE: — Talker and listener both share a single address of 03H.

CPU WRITES TO:	PATTERN	COMMENT
Address Mode Register	0000 0001	Selects Mode 1 Addressing
Address 0/1 Register	0000 0011	Talker and Listener Address = 03
Address 0/1 Register	1110 0000	Minor Address is disabled

4. MODE: — Talker and listener have a primary address of 04H and a secondary address of 05H

CPU WRITES TO:	PATTERN	COMMENT
Address Mode Register	0000 0010	Selects Mode 2 Addressing
Address 0/1 Register	0000 0100	Primary Address = 04H
Address 0/1 Register	1000 0101	Minor Address is disabled

5. MODE: — Talker has a primary address of 06H. Listener has a primary address of 07H

CPU WRITES TO:	PATTERN	COMMENT
Address Mode Register	0000 0011	Select Mode 3
Address 0/1 Register	0010 0110	Talker Address = 06
Address 0/1 Register	1100 0111	Listener Primary = 07

The CPU will verify the secondary addresses which could be the same or different.

APPLICATION OF THE 8291A

This phase of the application note will examine programming of the 8291A, corresponding bus commands and responses, CPU interruption, etc. for a variety of GPIB activities. This should provide the reader with a clear understanding of the role the 8291A performs in a GPIB system. The talker function, listener function, remote message handling, and remote/local operations including local lockout, are discussed.

Talker Functions

TALK-ONLY (ton). In talk only mode the 8291A will not respond to the MTA message from a controller. Generally, ton is used in an environment which does not have a controller. Ton is also employed in an interface that includes the controller function.

When the 8291A is used with the 8292, the sequence of events for initialization are as follows:

1) The Interrupt/Enable registers are programmed.
2) Ton is selected.
3) Settling time is selected.
4) EOS character is loaded.
5) "Pon" local message is sent.
6) CPU waits for Byte Out (BO) and sends a byte to the data out register.

Addressed Talker (Via MTA Message)

The GPIB controller will direct the 8291A to talk by sending a My Talk Address (MTA) message containing the 8291A's talk address. The sequence of events is as follows:

1) The interrupt enable and serial poll mode registers are programmed.
2) Mode 1 is selected.
3) Settling time is selected.
4) Talker and listener addresses are programmed.
5) Power on (pon) local message is sent.
6) CPU waits for an interrupt. When the controller has sent the MTA message for the 8291A an interrupt will be generated if enabled and the ADSC bit will be set.
7) CPU reads the Address Status register to determine if the 8291A has been addressed to talk (TA = 1).
8) CPU waits for an interrupt from either BO or ADSC.
9) When BO is set, the CPU writes the data byte to the data out register.
10) CPU continues to poll the status registers.
11) When unaddressed ADSC, will be set and TA reset.

LISTENER FUNCTIONS

LISTEN-ONLY (lon). In listen-only mode the 8291A will not respond to the My Listen Address (MLA) message from the controller. The sequence of events is as follows:

1) The Interrupt Enable registers are programmed.
2) Lon is selected.
3) EOS characted is programmed.
4) "Pon" local message is sent.
5) CPU waits for BI and reads the byte from the data-in register.

Note that enabling both ton and lon can create an internal loopback as long as another listener exists.

Addressed Listening (Via the MLA Message)

The GPIB controller will direct the 8291A to listen by sending a MLA message containing the 8291A's listen address. The sequence of events is as follows:

1) The Interrupt Enable registers are programmed.
2) The serial poll mode register is loaded as desired.
3) Talker and listener addresses are loaded.
4) "Pon" local message is sent
5) The CPU waits for an interrupt. When the controller has sent the MLA message for the 8291A, the ADSC bit will be set.
6) The CPU reads the Address Status Register to determine if the 8291A has been addressed to listen (LA = 1).
7) CPU waits for an interrupt for BI or ADSC.
8) When BI is set, the CPU reads the data byte from the data-in register.
9) The CPU continues to poll the status registers.
10) When unaddressed, ADSC will be set and LA reset.

Remote/Local and Lockout

Remote and local refer to the source of control of a device connected to the GPIB. Remote refers to control from the GPIB controller-in-charge. Local refers to control from the device's own system. Reference should be made to the RL state diagram in the 8291A data sheet.

Upon "pon" the 8291A is in the local state. In this state the REM bit in Interrupt Status 1 Register is reset. When the GPIB controller takes control of the bus it will drive the REN (remote enable) line true. This will cause the REM bit and REMC (remote/local change) bit to be set. The distinction between remote and local modes is necessary in that some types of devices will have local controls which have functions which are also controlled by remote messages.

In the local state the device is allowed to store, but not respond to, remote messages which control functions which are also controlled by local messages. A device

which has been addressed to listen will exit the local state and go the the remote state if the REN message is true and the local rtl (return to local) message is false. The state of the "rtl" local message is ignored and the device is "locked" into the local state if the LLO remote message is true. In the Remote state the device is not allowed to respond to local message which control function that are also controlled by remote messages. A device wil exit the remote state and enter the local state when REN goes false. It will also enter the local state if the GTL (go to local) remote message is true and the device has been addressed to listen. It will also enter the local state if the rtl message is true and the LLO message is false or ACDS is inactive.

A device will exit the remote state and enter RWLS (remote with lockout state) if the LLO (local lockout) message is true and ACDS is active. In this mode, those local message which control functions which are also controlled by remote messages are ignored. In other words, the "rtl": message is ignored. A device will exit RWLS and to to the local state if REN goes false. The device will exit RWLS and go to LWLS if the GTL message is true and the device is addressed to listen.

Polling

The IEEE-488 standard specifies two methods for a slave device to let the controller know that it needs service.

These two methods are called Serial and Parallel Poll. The controller performs one of these two polling methods. after a slave device requests service. As implied in the name, a Serial Poll is when the controller sequentially asks each device if it requested service. In a Parallel Poll the controller asks all of the devices on the GPIB if they requested service, and they reply in parallel.

Serial Poll

When the controller performs a Serial Poll, each slave device sends back to the controller a Serial Poll Status Byte. One of the bits in the Serial Poll Status Byte indicates whether this device requested service or not. The remaining 7 bits are user defined, and they are used to indicate what type of service is required. The IEEE-488 spec only defines the service request bit, however HP has defined a few more bits in the Serial Poll Status Byte. This can be seen in figure 4.

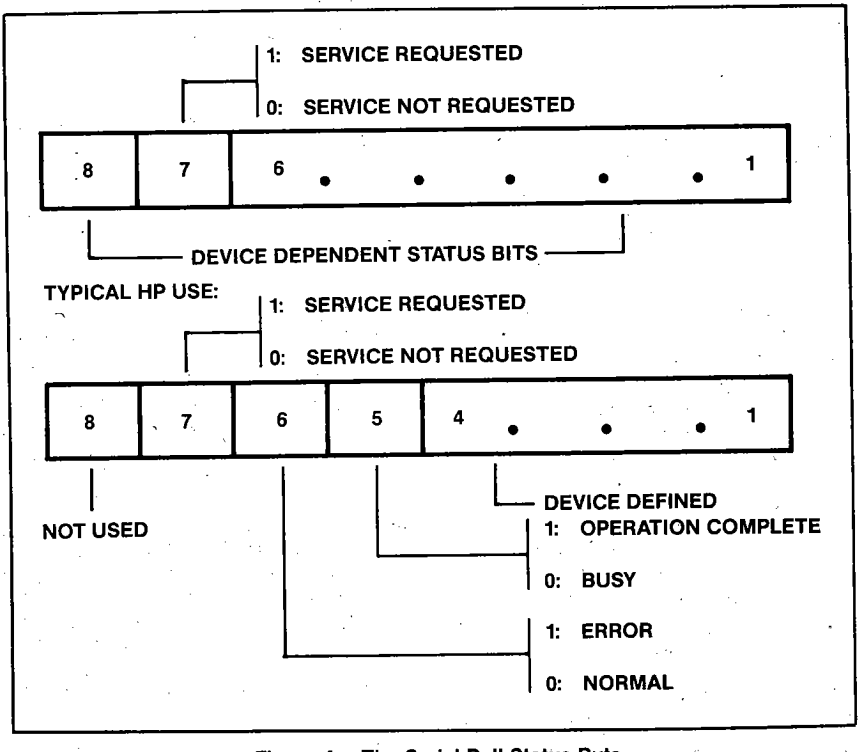


Figure 4. The Serial Poll Status Byte

When a slave device needs service it drives the SRQ line on the GPIB bus true (low). For the 8291A this is done by setting bit 7 in the Serial Poll Status Byte. The CPU in the controller may be interrupted by SRQ or it may poll a register to determine the state of SRQ. Using the 8292 one could either poll the interrupt status register for the SRQ interrupt status bit, or enable SRQ to interrupt the CPU. After the controller recognizes a service request, it goes into the serial poll routine.

The first thing the controller does in the serial poll routine is assert ATN. When ATN is asserted true the controller takes control of the GPIB, and all slave devices on the bus must listen. All bytes sent over the bus while ATN is true are commands. After the controller takes control, it sends out a Universal Unlisten (UNL), which tells all previously addressed listeners to stop listening. The controller then sends out a byte called SPE (Serial Poll Enable). This command notifies all of the slaves on the bus that the controller has put the GPIB in the Serial Poll Mode State (SPMS). Now the controller addresses the first slave device to TALK and puts itself in the listen mode. When the controller resets ATN the device addressed to talk transmits to the controller its Serial Poll Status Byte. If the device just polled was the one requesting service, the SRQ line on the GPIB goes false, and bit 7 in the serial poll status byte of the 8291A is reset. If more than one device is requesting service, SRQ remains low until all of the devices requesting service have been polled, since SRQ is wire-ored. To continue the Serial Poll, the controller asserts ATN, addresses the next device to talk then reads the Serial Poll Status Byte. When the controller is finished polling it asserts ATN, sends the universal untalk command (UNT), then sends the Serial Poll Disable command (SPD). The flow of the serial poll can be seen from the example in figure 5.

0. DEVICE A REQUESTS SERVICE (SRQ)
1. ASSERT ATN
2. UNIVERSAL UNLISTEN (UNL)
3. SERIAL POLL ENABLE (SPE)
4. DEVICE A TALK ADDRESS (MTA)
5. RELEASE ATN
6. DEVICE A STATUS BYTE (STB) (RQS SET)
7. ASSERT ATN
8. DEVICE B TALK ADDRESS (MTA)
9. RELEASE ATN
10. DEVICE B STATUS BYTE (STB) (RQS CLEAR)
11. ASSERT ATN
12. DEVICE C TALK ADDRESS (MTA)
13. RELEASE ATN
14. DEVICE C STATUS BYTE (STB) (RQS CLEAR)
15. ASSERT ATN
16. UNIVESAL UNTALK (UNT)
17. SERIAL POLL DISABLE (SPD)
18. GO PROCESS SERVICE REQUEST

Figure 5. Serial Polling

The following section describes the events which happen in a serial poll when 8291A and 8292 are the controller, and another 8291A is the slave device. While going through this section the reader should refer to the register diagrams for the 8291A and 8292.

A. DEVICE A REQUESTS SERVICE (SRQ BECOMES TRUE)

The slave devices rsv bit in the 8291A's serial poll mode register is set.

B. CONTROLLER RECOGNIZES SRQ AND ASSERTS ATN

Thf 8292's SPI pin 33 interrupts the CPU. The CPU reads the 8292's Interrupt status register and finds the SRQ bit set. The CPU tells the 8292 to 'Take Control Synchronously' by writing a OFDH to the 8292's command register.

C. THE CONTROLLER SENDS OUT THE FOLLOWING COMMANDS: UNIVERSAL UNLISTEN (UNL), SERIAL POLL ENABLE (SPE), MY TALK ADDRESS (MTA).

(MTA is a command which tells one of the devices on the bus to talk.)

The CPU in the controller waits for a BO (byte out) interrupt in the 8291A's interrupt status 1 register before it writes to the Data Out register a 3FH (UNL), 18H (SPE), 010XXXXX (MTA). The X represents the programmable address of a device on the GPIB. When the 8291A in the slave device receives its talk address, the ADSC bit in the Interrupt Status register 2 is set, and in the Address Status Register TA and TPAS bits are set.

D. CONTROLLER RECONFIGURES ITSELF TO LISTEN AND RESETS ATN

The CPU in the controller puts the 8291A in the listen only mode by writing a 40H to the Address Mode register of the 8291A, and then a OOH to the Aux Mode register. The second write is an 'Immediate Execute pon' which must be used when switching addressing modes such as talk only to listen only. To reset ATN the CPU tells the 8292 to 'Go To Standby' by writing a 0F6H to the command register. The moment ATN is reset, the 8291A in the slave device sets SPAS in Interrupt Status 2 register, and transmits the serial poll status byte. SRQS in the Serial Poll Status byte of the 8291A slave device is reset, and the SRQ line on the GPIB bus becomes false.

E. THE CONTROLLER READS THE SERIAL POLL STATUS BYTE, SETS ATN, THEN RECONFIGURES ITSELF TO TALK

The CPU in the controller waits for the Byte In bit (BI) in the 8291A's Interrupt Status 1 register. When this bit is set the CPU reads the Data In register to receive the Serial Poll Status Byte. Since bit 7 is set, this was the device which requested service. The CPU in the controller tells the 8292 to 'Take Control Synchronously' which asserts ATN. The moment ATN is asserted true the 8291A in the slave device resets SPAS, and sets the Serial Poll Com-

plete (SPC) bit in the Interrupt Status 2 register. The controller reconfigures itself to talk by setting the TO bit in the Address Mode register and then writing a OOH to the Aux Mode register.

F. THE CONTROLLER SENDS THE COMMANDS UNIVERSAL UNTALK (UNT), AND SERIAL POLL DISABLE (SPD) THEN RESETS THE SRQ BIT IN THE 8292 INTERRUPT STATUS REGISTER

The CPU in the controller waits for the BO Interrupt status bit to be set in the Interrupt Status 1 register of the 8291A before it writes 5FH (UNT) and 19H (SPD) to the Data Out register. The CPU then writes a 2BH to the 8292's command register to reset the SRQ status bit in the Interrupt Status register. When the 8291A in the slave device receives the UNT command the ADSC bit in the Interrupt Status 2 register is set, and the TA and TPAS bits in the Address Status register will be reset. At this point the controller can service the slave device's request.

Note that in the software listing of AP-66 (USING THE 8292 GPIB CONTROLLER) there is a bug in the serial poll routines. In the 'SRQ ROUTINE' when the CPU finds that the SRQ bit in the interrupt status register is set, it immediately writes the interrupt Acknowledge command to the 8292 to reset this bit. However the SRQ GPIB line will still be driven true until the slave device driving SRQ has been polled. Therefore, the SRQ status bit in the 8292 will become set and latched again, and as a result the SRQ status bit in the 8292 will still be set after the serial poll. The proper time to reset the SRQ bit in the 8292 is after SRQ on the GPIB becomes false.

Parallel Poll

The 8291A supports an additional method for obtaining status from devices known as parallel poll (PPOL). This method limits the controller to a maximum of 8 devices at a time since each device will produce a single bit response on the GPIB data lines. As shown in the state diagrams, there are three basic parallel poll sates: PPIS (parallel poll idle state), PPSS (parallel poll standby state), and PPAS (parallel poll active state).

In PPIS, the device's parallel poll function is in the idle state and will not respond to a parallel poll. PPSS is the standby state, a state in which the device will respond to a parallel poll from the controller. The response is inititated by the controller driving both ATN and EOI true simultaneously.

The 8291A state diagram shows a transition from PPIS to PPSS with the "lpe" message. This is a PP2 implementation for a parallel poll. This "lpe" (local poll enable) local message is achieved by writing $011USP_3P_2P_1$ to the Aux Mode Register with U=0. The S bit is the sense bit. If the "ist" (individual status) local message value matches the sense bit, then the 8291A will give a true response to a parallel poll. Bits P_3-P_1 identify which data line is used for a response.

For example, assume the programmer decides that the system containing the 8291A shall participate in parallel poll. The programmer, upon system initialization would write to the Aux Mode Register and reset the U bit and set the S bit plus identify a data line (P_3-P_1 bits). At "pon," the 8291A would not resond true to a parallel poll unless the parallel poll flag is set (via Aux Mode Register command).

When a status condition in the user system occurs and the programmer decides that this condition warrants a true response, then programmers software should set the parallel poll flag. Since the S bit value matches the "ist" (set) condition,a true response will be given to all parallel polls.

An additional method of parallel polling reading exists known as a PP1 implementation. In this case the controller sends a PPE (parallel poll enable) message. PPE contains a bit pattern similar to the bit pattern used to program the "lpe" local message. The 8291A will receive this as an undefined command and use it to generate an "lpe" message. Thus the controller is specifying the sense bits and data lines for a response. A PPD (parallel poll disable) message exists which clears the bits $SP_3P_2P_1$ and sets the U bit. This also will be received by the 8291A and used to generate an "lpe" false local message.

The actual sequence of events is as follows. The controller sends a PPC (parallel poll configure) message. This is an undefined command which is received in the CPT register and the handshake is held off. The local CPU reads this bit pattern, decodes it, and sends a VSCMD message to the Aux Mode Register. The controller then sends a ppe message which is also recieved as a undefined command in the CPT register. The local CPU reads this, decodes it clears the MSB, and writes this to the Aux Mode Register generating the "lpe" message.

The controller then sends ATN and EOI true and the 8291A drives the appropriate data line if the "ist" (parallel poll flag) is true. The controller will then send a PPD (parallel poll disable) message (again, an undefined command). The CPU reads this from the CPT register and uses it to write a new "lpe" message (this "lpe" message will be false). The controller then sends a PPU (parallel poll unconfigure) message. Since this is also an undefined command, it goes into the CPT register. When the local CPU decodes this, the CPU should clear the "ist" (parallel poll flag).

APPLICATION EXAMPLES

In the course of developing this application note, two complete and identical GPIB systems were built. The

schematics and block diagrams are contained in Appendix 1. These systems feature an 8088 CPU, 8237 DMA controller, serial I/O (8251A and 8253), RAM, EPROM, and a complete GPIB talker/listener controller. Jumper switches were provided to select between a controller function and a talker/listener function. This system design is based on the design of Intel's SDK-86 prototyping kit and thus shares the same I/O and memory addresses. This system uses the same download software to transfer object files from Intel development systems.

Two Software Drivers

Two software drivers were developed to demonstrate a ton/lon environment. These two programs (BOARD 1 and BOARD 2) are contained in Appendix 2.

In this example, one of the systems (BOARD 1) initially is programmed in talk-only mode and synchronization is achieved by waiting for the listening board to become active. This is sensed by the lack of a GPIB error since a condition of no active listener produces an ERR status condition. Board 1 upon detecting the presence of an active listener transmitts a block of 100 bytes from a PROM memory across the bus. The second system (BOARD 2) receives this data and stores it in a buffer, EOI is sent true by the talker (BOARD 1) with the last byte of data. Upon detection of EOI, BOARD 2 switches to the talk only mode while BOARD 1 upon terminal count switches to the listen only mode. BOARD 2 then detects the presence of an active listener and transmitts the contents of its buffer back to BOARD 1 which stores this data in the buffer. EOI again is sent with the last byte and BOARD 2 switches back to listen-only. BOARD 1 upon detecting EOI then compares the contents of its buffer with the contents of its PROM to ensure that no data transmission errors occured. The process then repeats itself.

8291A with HP 9835A

An example of the 8291A used in conjunction with a bus controller is also included in this application note. In this example, the 8291A system used in previous experiments was connected via the GPIB to a Hewlett-Packard 9835A desktop computer. This computer contains, in addition to a GPIB interface, a black and white CRT, keyboard, tape drive for high quality data cassettes, and a calculator type printer. The software for the HP 9835A is shown in Appendix 3. The user should refer to the operation manuals for the HP 9835A for information on the features and programming methods for the HP 9835A.

In this example, the 8292 was removed from its socket and the OPTA and OPTB pins of the two 8293 transceiver reconfigured to modes 0 and 1. Optionally, the mode pins could have been left wired for modes 2 and 3 and the 8292 left in its socket with its SYC pin wired to ground. This would have produced the same effect.

The first action performed is sending IFC. Generally, this is done when a controller first comes on line. This pulse is at least 100 us in duration as specified by the IEEE-488 standard.

The software checks to see if active listeners are on line. For demonstration purposes, the HP 9835A will flag the operator to indicate that listeners are on line.

The HP 9835A then configures and performs a parallel poll (PPOL). The parallel poll indicates 1 bit of status of each device in a group of up to 8 devices. Such information could be used by an application program to determine whether optional devices are part of a system configuration. Such optional devices might include mass storage devices, printers, etc. where the application software for the controller might need to format data to match each type of device. Once the PPOL sequence is finished, the HP 9835A offers the user the opportunity to execute user commands from the keyboard. At this time the HP 9835A sits in a loop waiting for an SRQ condition. When the operator hits a key on the keyboard, the HP 9835A processor is interrupted and vectors to a service routine where the key is read and the appropriate routine is executed. The HP 9835A will then return to the loop checking for SRQ true. For this application, the valid keys are G,D,R,H,and X. Pressing the "G" key causes the GET command to be sent across the bus. A message to this effect is printed in the CRT and the HP 9835A returns. The "D" key causes the SDC message to be sent with the 8291A being the addressed device. Again, an appropriate message is output on th HP 9835A CRT. The "R" key causes the GTL message to be sent. The CRT displays "REMOTE MESSAGE SENT." The "H" key causes a menu to be displayed on the HP 9835A CRT screen. This menu lists the allowed commands and their functions. NO GPIB commands are sent. The "X" key allows the operator to send one line of data across the bus. The line of data is terminated by a carriage return and line feed produced by pressing the "CONTINUE" key on the HP 9835A.

The characters are stored in the sequence entered into a buffer whose maximum size is 80 characters. Pressing the "CONTINUE" key terminates storing characters in the array and all characters including the carriage return and line feed are sent. EOI is then sent true with a false byte of OOH. This false byte is due to the 1975 standard which allows asyncronous sending and reception of EOI. (The 8291A supports the later 1978 standard which eliminates this false byte).

After any key command is serviced control returns to the loop which checks for SRQ active. Should SRQ be active, then the keyboard interrupt is disabled and a message printed to indicate that SRQ has been received true.

The controller then performs a parallel poll.

This is an example of how parallel poll may be used to

quickly check which group of devices contains a device sending SRQ. The eight devices in a group would, of course, have software drivers which allow a true response to a PPOL if that device is currently driving SRQ true. This would be a valuable method of isolation of the SRQ source in a system with a large number of devices. In this application program, only the response from the 8291A is of concern and only the 8291A's response is considered. It does, however, demonstrate the technique employed. If a true response from the 8291A is detected, then a message to this effect is printed on the HP 9835A CRT screen. From this process, the controller has identified the device requesting service and will use a serial poll(SPOL) to determine the reason for the service request. This method of using PPOL is not specifically defined by the IEEE-488 standard but is a use of the resources provided.

The controller software then prints a message to indicate that it is about to perform a serial poll. This serial poll will return to the controller the current status of the 8291A and clear the service request. The status byte received is then printed on the CRT screen of the HP 9835A. One of the 8291A status bits indicates that the 8291A system has a field (on line or less) of data to transfer to the HP 9835A. If this bit is set, then the HP 9835A addresses the 8291A system to talk. The data is sent by the 8291A system is then printed on the CRT screen of the HP 9835A. The HP 9835 then enables the keyboard interrupts and goes into its SRQ checking loop.

Appendix 4 contains the software for the 8291A system which is connected to the HP 9835A via the GPIB. This software throws away the first byte of data it receives since this transfer was used by the HP 9835A to test when the 8291A system came on line.

Next, both status registers are read and stored in the two variable STAT 1 and STAT 2. It is necessary to store the status since reading the status registers clears the status bits.

Initially, six status bits are evaluated (END, GET, CPT, DEC, REMC, ADSC). Some of these conditions require that additional status bits be evaluated.

If END is true, then the 8291A system has received a block from the HP 9835A and the contents of a buffer is printed on the CRT screen. Next, the CPT bit is checked. PPC and PPE are the only valid undefined commands in this example.

Next, the GET bit is examined and if true, the CRT screen connected to the serial channel on the 8291A system prints a message to indicate that the trigger command has been received. A similar process occurs with the DEC and REMC status bits.

Address Status Change (ADSC) is checked to see if the 8291A has been addressed or unaddressed by the controller. If ADSC is false, then the software checks the keyboard at the CRT terminal. If ADSC is set, then the TA and LA bits are read and evaluated to determine whether the 8291A has been addressed to talk or listen. The DMA controller is set to start transfers at the start of the character buffer and the type of transfer is determined by whether the 8291A is in TADS or LADS. We only need to set up the DMA controller since the transfers will be transparent to the system processor. The keyboard from the CRT terminal is then checked. If a key as been hit, then this character is stored in the character buffer and the buffer printer set to the next character location. This process repeats until the received character is a line feed. The line feed is echoed to the CRT, the serial poll status byte updated and the SRQ line driven true. This allows the 8291A system to store up to one line of characters before requesting a transfer to the controller. Recall that upon receiving an SRQ, the controller will perform a serial poll and subsequently address the 8291A to talk. The 8291A system then goes back to reading the status register thus repeating the process.

CONCLUSION

This application note has shown a basic method to view the IEEE 488 bus, when used in conjunction with Intel's® 8291A.

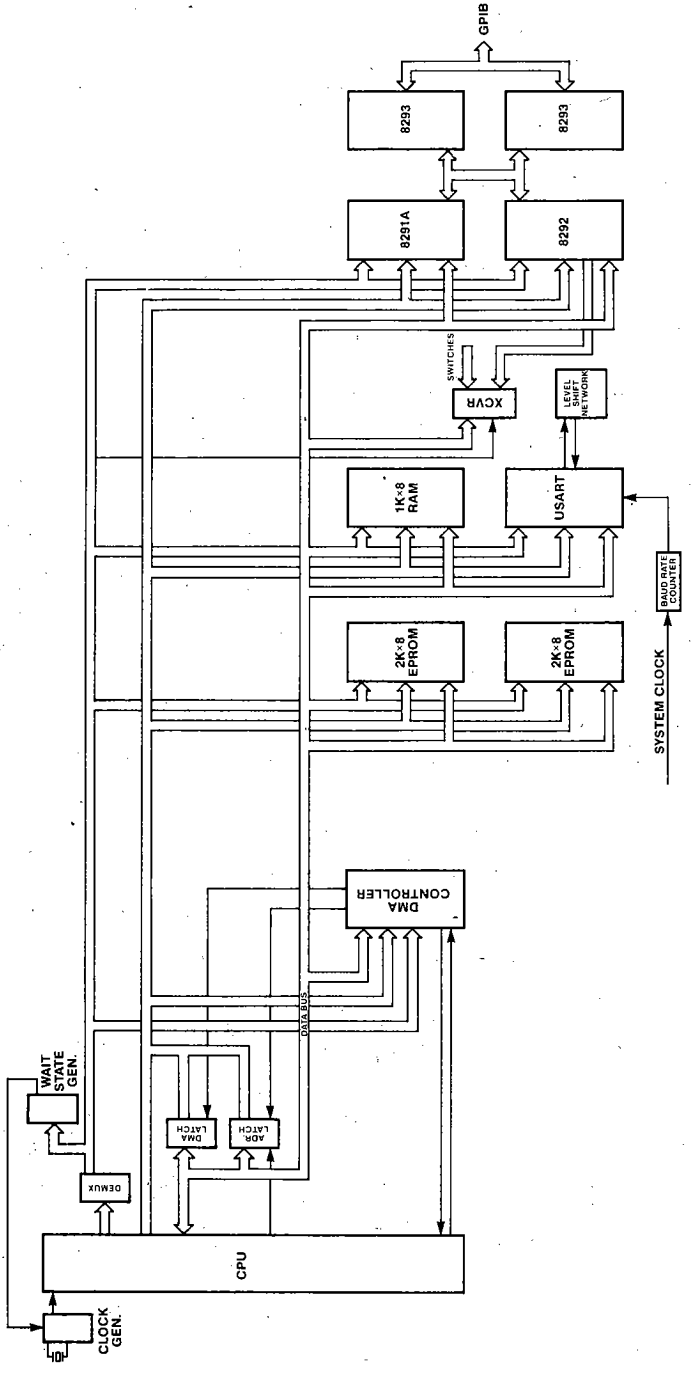
The main reference for GPIB questions is the IEEE Standard 488 - 1978. Reference 8291A's data sheet for detailed information on it.

Additional Intel® GPIB products include iSBX-488, which is a multimode board consisting of the 8291A, 8292, and 8293.

REFERENCES

8291A Data Sheet
8292 Data Sheet
8293 Data Sheet
Application Note #66 "Using the 8292 GPIB Controller"
PLM-86 User Manual
HP 9835A User's Manual
IEEE—488—1978 Standard

APPENDIX 1
SYSTEM BLOCK DIAGRAM WITH 8088



APPENDIX 2
SOFTWARE DRIVERS FOR BLOCK DATA TRANSFER

```
PL/M-86 COMPILER    BOARD 1

ISIS-II PL/M-86 V1.1 COMPILATION OF MODULE BOARD 1
OBJECT MODULE PLACED IN:  F1.  BRD1   OBJ
COMPILER INVOKED BY:    PLM86.  F1:  BRD1.  SRC SYMBOLS MEDIUM

          /*   BOARD 1 TPT PROGRAM        */
          /*   THIS BOARD TALKS TO THE OTHER BOARD BY  */
          /*   TRANSFERRING A BLOCK OF DATA VIA THE 8237  */
          /*   COUPLED WITH THE 8291A.    THE 8291A IS PROGRAM- */
          /*   MED TO SEND EOI WHEN RECOGNIZING THE LAST     */
          /*   DATA BYTE'S BIT PATTERN.  WHILE DATA IS BEING */
          /*   TRANSFERRED, THE PROCESSOR PERFORMS  I/O READS */
          /*   OF THE 8237 COUNT REGISTERS TO SIMULATE BUS    */
          /*   ACTIVITY, AND TO DETERMINE WHEN TO TURN THE    */
          /*   LINE AROUND.  AFTER THE 8237 HAS REACHED  */
          /*   TERMINAL COUNT, THE 8291A IS PROGRAMMED TO      */
          /*   THE LISTENER STATE AND WAITS FOR THE BLOCK    */
          /*   TO BE TRANSMITTED BACK FROM THE SECOND BOARD.  */
          /*   THIS DATA IS PLACED IN A SECOND BUFFER AND     */
          /*   ITS CONTENTS COMPARED WITH THE ORIGINAL DATA   */
          /*   TO CHECK FOR INTERFACE INTEGRITY.   */

1         BOARD1:

          DO;
               /* PROCEDURES */

2    1         CO:  PROCEDURE (XXX) ;
3    2              DECLARE XXX BYTE,
                    SER$STAT  LITERALLY  'OFFF2H',
                    SER$DATA  LITERALLY  'OFFF0H',
                    TXRDY     LITERALLY  '01H',
4    2              DO WHILE (INPUT  (SER$STAT) AND TXRDY) <>     TXRDY;
5    3              END;
6    2              OUTPUT (SER$DATA)      = XXX;
7    2         END CO;

                    /*   SETUP BUFFERS */
8    1              DECLARE BUFF2 (100)  BYTE;    /*  RAM STORAGE AREA */
9    1                   DECLARE BUFF1 (100) BYTE DATA

                    (1,2,3,4,5,6,7,8,9,10H,
                    11H, 12H, 13H, 14H, 15H, 16H, 17H, 18H, 19H, 20H,
                    21H, 22H, 23H, 24H, 25H, 26H, 27H, 28H, 29H, 30H,
                    31H, 32H, 33H, 34H, 35H, 36H, 37H, 38H, 39H, 40H,
                    41H, 42H, 43H, 44H, 45H, 46H, 47H, 48H, 49H, 50H,
                    51H, 52H, 53H, 54H, 55H, 56H, 57H, 58H, 59H, 60H,
                    61H, 62H, 63H, 64H, 65H, 66H, 67H, 68H, 69H, 70H,
                    71H, 72H, 73H, 74H, 75H, 76H, 77H, 78H, 79H, 80H,
                    81H, 82H, 83H, 84H, 85H, 86H, 87H, 88H, 89H, 90H,
```

```
PL/M-86  COMPILER  BOARD1

                    91H, 92H, 93H, 94H, 95H, 96H, 97H, 98H, 99H, 0DH);
10  1               DECLARE BUFF3(17)  BYTE DATA
                    (0DH, 0AH, 'COMPARE ERROR', 0DH, 0AH);   /* ROM STORAGE AREA */

                    /* 8237 PORT ADDRESSES  */

11  1               DECLARE

                    CLEAR$FF        LITERALLY '0FFDDH', /* MASTER CLEAR */
                    START$0$LO      LITERALLY '0FFD0H',
                    START$0$HI       LITERALLY '0FFD0H',
                    0$COUNT$LO      LITERALLY '0FFD1H',
                    0$COUNT$HI      LITERALLY '0FFD1H',
                    SET$MODE        LITERALLY '0FFDBH',
                    CMD$37          LITERALLY '0FFD8H',
                    SET$MASK        LITERALLY '0FFDFH';

                    /* 8237 COMMAND  - DATA BYTES */
12  1               DECLARE         DMA$ADR$TALK   POINTER;
13  1               DECLARE         DMA$ADR$LSTN .  POINTER;

14  1               DECLARE

                    RD$TRANSFER     LITERALLY '48H',
                    WR$TRANSFER     LITERALLY '44H',
                    NORM$TIME       LITERALLY '20H',
                    TC$LO1          LITERALLY '0FFH',
                    TC$HI1          LITERALLY '00H',
                    TC$LO2          LITERALLY '99D',          /* 100 XFERS  */
                    TC          LITERALLY '01H',                          \
                    I               BYTE;

15  1               DECLARE

                    DMA$WRD$TALK (2)  WORD   AT   (@DMA$ADR$TALK),
                    DMA$WRD$LSTN(2)  WORD    AT   (@DMA$ADR$LSTN);

                    /* 8291A PORT ADDRESSES */

16  1               DECLARE

                    PORT$OUT        LITERALLY '0FFC0H', /* DATA OUT*/
                    PORT$IN         LITERALLY '0FFC0H'
                    STATUS$1        LITERALLY '0FFC1H', /*INTR STAT 2*/
                    STATUS$2        LITERALLY '0FFC2H', /* INTR STAT 2 */
                    ADDR$STATUS     LITERALLY '0FFC4H',
                    COMMAND$MOD     LITERALLY '0FFC5H', /*CMD PASS THRU */
                    ADDR$0          LITERALLY '0FFC6H',
                    EOS$REG         LITERALLY '0FFC7H', /*  EOS REGISTER */
```

```
                /* 8291A COMMAND  - DATA BYTES */

PL/M-86 COMPILER    BOARD1

17  1                   DECLARE

                    END$EOI    LITERALLY        '88H',
                    DNE   LITERALLY        '10H',
                    PON   LITERALLY        '00H',
                    RESET      LITERALLY        '02H',
                    CLEAR      LITERALLY        '00H',
                    DMA$REQ$L LITERALLY        '10H',
                    DMA$REQ$T LITERALLY        '20H',
                    MOD1$TO    LITERALLY        '80H',
                    MOD1$LO    LITERALLY        '40H',
                    EOS   LITERALLY        '0DH',
                    PRESCALER LITERALLY        '23H',
                    HIGH$SPEED LITERALLY            '0A4H',
                    OKAY       LITERALLY        'OFFFFH',
                    XYZ   BYTE,
                    MATCH      WORD,
                    BO    LITERALLY        '02H',
                    BI    LITERALLY        '01H',
                    ERR   LITERALLY        '04H';

          /* CODE BEGINS */

18  1     START91:

                    OUTPUT  (STATUS$2)  =CLEAR;  /* SHUT-OFF DMA REQ BITS TO */
                                                 /* PREVENT EXTRA DMA REQS  */
                                                  /*FROM 8291A                */

                    /* MANIPULATE DMA ADDRESS VARIABLES */

19  1               DMA$ADR$TALK   =(@BUFF1);
20  1               DMA$ADR$LSTN   =(@BUFF2);
21  1               DMA$WRD$TALK(1)=SHL (DMA$WRD$TALK(1), 4);
22  1               DMA$WRD$TALK(0)=DMA$WRD$TALK (0) + DMA$WRD$TALK (1);
23  1               DMA$WRD$LSTN(1)=SHL (DMA$WRD$LSTN (1), 4);
24  1               DMA$WRD$LSTN(0)=DMA$WRD$LSTN (0) +DMA$WRD$LSTN (1);

25  1          INIT37T:
                /* INIT 8237 FOR TALKER FUNCTIONS */

26  1               OUTPUT     (CLEAR$FF)      =CLEAR;/* TOGGLE MASTER CLEAR */
27  1               OUTPUT     (CMD$37)        =NORM$TIME;
28  1               OUTPUT     (SET$MODE)      =RD$TRANSFER;
                    OUTPUT     (SET$MASK)      =CLEAR;
29  1               OUTPUT     (START$0$LO)       =DMA$WRD$TALK (0);
30  1               DMA$WRD$TALK (0)           =SHR (DMA$WRD$TALK (0), 8);
31  1               OUTPUT     (START$0$HI)       =DMA$WRD$TALK (0);
32  1               OUTPUT     (0$COUNT$LO)       =TC$LO2;
33  1               OUTPUT     (0$COUNT$HI)       =TC$HI2;
                    /* INIT 8291A FOR TALKER FUNCTIONS */

PL/M-86 COMPILER      BOARD1
```

```
34  1               OUTPUT  (EOS$REG)     =EOS;
35  1               OUTPUT  (COMMAND$MOD)     =END$EOI; /*  EOI ON EOS SENT */
36  1               OUTPUT  (ADDR$STATUS)     =MOD1$TO; /* TALK ONLY */
37  1               OUTPUT  (COMMAND$MOD)     =PRESCALER;
38  1               OUTPUT  (COMMAND$MOD)     =HIGH$SPEED;
39  1               OUTPUT  (COMMAND$MOD)     =PON;

40  1               DO WHILE (INPUT  (STATUS$1) AND BO)  =0;
41  2               END;     /*  WAIT FOR BO INTR  */
42  1               OUTPUT  (PORT$OUT)  = 0AAH;

43  1               DO WHILE  (INPUT  (STATUS$1) AND ERR) = ERR;
44  2                  DO WHILE  (INPUT  (STATUS$1)  AND BO) = 0;
45  3                  END;  /*  WAIT FOR BO INTR  */
46  2                  OUTPUT  (PORT$OUT)  =0AAH;
47  2               END;

48  1               OUTPUT  (STATUS$2)  =DMA$REQ$T;  /*  ENABLE DMA REQS */

49  1                   DO WHILE  (INPUT  (CMD$37)  AND TC) <>         TC;
                     /* WAIT FOR TC  = 0 */
50  2                   END;

51  1               INIT37L;

                    OUTPUT  (STATUS$2)  =CLEAR;  /*  DISABLE DMA REQS */

                    /* INIT 8237 FOR LISTENER FUNCTIONS */

52  1               OUTPUT  (CLEAR$FF) 0=CLEAR; /*  TOGGLE MASTER RESET */
53  2               OUTPUT  (CMD$37)     =NORM$TIME;
54  1               OUTPUT  (SET$MODE)  =WR$TRANSFER;
55  1               OUTPUT  (SET$MASK)  =CLEAR;
56  1               OUTPUT  (START$0$LO)    =DMA$WRD$LSTN (0);
57  1               DMA$WRD$LSTN (0)     =SHR (DMA$WRD$LSTN (0), 8);
58  1
    OUTPUT      (START $0$HI)      =DMA$WRD$LSTN (0);
59  1               OUTPUT  (0$COUNT$LO)      =TC$LO1;
60  1               OUTPUT  (0$COUNT$HI)      =TC$HI1;

                    /* INIT 8291A FOR LISTENER FUNCTIONS  */

61  1               OUTPUT  (COMMAND$MOD)     =RESET;
62  1               OUTPUT  (ADDR$STATUS)     =MOD1$LO; /*  LISTEN ONLY */
63  1               OUTPUT  (COMMAND$MOD)     =PON;

64  1               DO WHILE (INPUT  (STATUS$1)  AND BI)  =0;
65  2               END;  /*  WAIT FOR BI INTR */
66  1               XYZ  = INPUT (PORT$IN);

67  1               OUTPUT  (STATUS$2)  =DMA$REQ$L;  /*  ENABLE DMA REQS  */

68  1               DO WHILE  (INPUT  (STATUS$1)  AND DNE)<>          DNE;
                    /*  WAIT FOR EOI RECEIVED  */
```

```
PL/M-86 COMPILER    BOARD 1

      70    1     CMPBLKS

                  /* COMPARE THE TWO BUFFERS CONTENTS */

                  MATCH=CMPB  (@BUFF1, @BUFF2, 100);

      71    1     IF MATCH = OKAY THEN GOTO START91;

                  /*  SEND ERROR MESSAGE IN BUFFER 3  */

      73    1     DO I=0 TO 16;
      74    2        CALL CO  (BUFF 3  (I)  );
      75    2     END;

      76    1     GOTO START91;

      77    1     END;


MODULE INFORMATION:

          CODE AREA SIZE          =01DBH    475D
          CONSTANT AREA SIZE      =0075H    117D
          VARIABLE AREA SIZE      =0070H    112D
          MAXIMUM STACK SIZE      =0006H      6D
          243 LINES READ
          0 PROGRAM ERROR (S)


END OF PL/M-86  COMPILATION
```

```
PL/M-86 COMPILER    BOARD2

ISIS-II PL/M-86  V1.1 COMPILATION OF MODULE BOARD2
OBJECT MODULE PLACED IN : F1: BRD2, OBJ
COMPILER INVOKED BY:     PLM86  :F1:  BRD2, SRC


                    /* BOARD 2 TPT PROGRAM        */
                    /*                            */
                    /* THIS BOARD LISTENS TO THE OTHER BOARD (1)   */
                    /* AND DMA'S DATA INTO A BUFFER, WHILE WAITING */
                    /* FOR THE END INTERRUPT BIT TO BECOME ACTIVE  */
                    /* UPON END ACTIVE, THE DATA IN THE BUFFER IS  */
                    /* SENT BACK TO THE FIRST BOARD VIA THE GPID   */
                    /* WHEN THE BLOCK IS FINISHED THE 8291A IS  */
                    /* PROGRAMMED BACK INTO THE LISTENER MODE    */

    1          BOARD2:

               DO;
                    /* 8237 PORT ADDRESSES */

2   1                DECLARE

                     CLEAR$FF        LITERALLY       'OFFDDH',     /*MASTER CLEAR */
                     START$O$Lo      LITERALLY       'OFFDOH',
                     START$O$HI      LITERALLY       'OFFDOH',
                     O$COUNT$LO      LITERALLY       'OFFD1H',
                     O$COUNT$HI      LITERALLY       'OFFD1H',
                     SET$MODE        LITERALLY       'OFFDBH',
                     CMD$37          LITERALLY       'OFFD8H',
                     SET$MASK        LITERALLY       'OFFDFH',

                     /* 8237 COMMAND  - DATA BYTES  */

3   1                DECLARE

                     RD$TRANSFER     LITERALLY       '48H',
                     WR$TRANSFER     LITERALLY       '44H',
                     ADDR$1A         LITERALLY       '00H',
                     ADDR$1B         LITERALLY       '01H',
                     NORM$TIME       LITERALLY       '20H',
                     TC$LO1          LITERALLY       'OFFH',
                     TC$HI1          LITERALLY       '00H',
                     TC$LO2          LITERALLY       '99D',
                     TC$HI2          LITERALLY       '00H',
                     TC         LITERALLY       '01H',

                     /* 8291A  PORT ADDRESSES */

4   1                DECLARE

                     PORT$OUT        LITERALLY       'OFFC0H',
                     PORT$IN         LITERALLY       'OFFC0H', /* DATA IN */
                     STATUS$1        LITERALLY       'OFFC1H',  /* INTR STAT 1 */
                     STATUS$2        LITERALLY       'OFFC2H',  /* INTR STAT 2 */
                     ADDR$STATUS     LITERALLY       'OFFC4H',  /* ADDR STAT   */
                     COMMAND$MOD     LITERALLY       'OFFC5H',  /* CMD PASS THRU */
```

```
PL/M-86 COMPILER   BOARD2

                        ADDR$0      LITERALLY       'OFFC6H',
                        EOS$REG     LITERALLY       'OFFC7H', /*  EOS REGISTER */

                        /* 8291A  COMMAND - DATA BYTES */

5   1                   DECLARE

                        END$EOI     LITERALLY       '88H',
                        DNE   LITERALLY       '10H',
                        PON   LITERALLY       '00H',
                        RESET       LITERALLY       '02H',
                        CLEAR       LITERALLY       '00H',
                        DMA$REQ$L   LITERALLY       '10H',
                        DMA$REQ$T   LITERALLY       '20H',
                        MOD1$TO     LITERALLY       '80H',
                        MOD1$LO     LITERALLY       '40',
                        EOS   LITERALLY       'ODH',
                        PRESCALER   LITERALLY       '23H',
                        HIGH$SPEED  LITERALLY       'A4H',
                        XYZ         BYTE,
                        BO          LITERALLY '02H',
                        BI          LITERALLY '01H',
                        ERR         LITERALLY '04H',

6   1               START91;

                        OUTPUT   (STATUS$2)   =CLEAR;   /*  END INITILIZATION STATE */

                        /* INIT 8237 FOR LISTENER FUNCTION  */

7   1               INIT37L;

                        OUTPUT   (CLEAR$FF)   =CLEAR; /*  TOGGLE MASTER RESET  */
8   1                   OUTPUT   (CMD$37)     =NORM$TIME;
9   1                   OUTPUT   (SET$MODE)   =WR$TRANSFER;   /*  BLOCK XFER MODE  */
10  1                   OUTPUT   (SET$MASK)   =CLEAR;
11  1                   OUTPUT   (START$0$LO)      =ADDR$1A;
12  1                   OUTPUT   (START$0$HI)      =ADDR$1B;
13  1                   OUTPUT   (0$COUNT$LO)      =TC$LO1;
14  1                   OUTPLUT  (0$COUNT$HI)      =TC$HI1;

                        /*  INIT 8291A  FOR LISTENER FUNCTIONS   */

15  1                   OUTPUT   (COMMAND$MOD)     =RESET;
16  1                   OUTPUT   (ADDR$STATUS)     =MOD1$LO;
17  1                   OUTPUT   (COMMAND$MOD)     =PON;
18  1                   DO WHILE (INPUT   (STATUS$1)  AND BI)  =0;
19  2                   END;   /*  WAIT FOR BI INTR */
20  1                   XYZ= INPUT   (PORT$IN);
21  1                   OUTPUT    (STATUS$2)      =DMA$REQ$L;

                        /* WAIT UNTIL EOI RCVD AND END INTR-BIT SET  */

22  1                      DO WHILE  (INPUT  (STATUS$1)  AND DNE ) <>   DNE;
```

```
PL/M-86  COMPILER  BOARD2

23  1                END;

24  1           INIT37T;
                  /* INIT 8237 FOR TALKER FUNCTION  */

                OUTPUT  (STATUS$2)  =CLEAR;    /* CLEAR  8291A  DRQ  */
25  1           OUTPUT  (CLEAR$FF)  =CLEAR;
26  1           OUTPUT  (CMD$37)    =NORM$TIME;
27  1           OUTPUT  (SET$MODE)  =RD$TRANSFER;   /*  BLOCK XFER MODE */
28  1           OUTPUT  (SET$MASK)  =CLEAR;
29  1           OUTPUT  (START$0$LO)      =ADDR$1A;
30  1           OUTPUT  (START$0$HI)      =ADDR$1B;
31  1           OUTPUT  (0$COUNT$LO)      =TC$LO2;
32  1           OUTPUT  (0$COUNT$HI)      =TC$HI2;

                /*  INIT 8291A  FOR TALKER FUNCTION  */

33  1           OUTPUT  (EOS$REG)    =EOS;
34  1           OUTPUT  (COMMAND$MOD)        =END$EOI; /* EOI ON EOS SENT  */
35  1           OUTPUT  (ADDR$STATUS)        =MOD1$TO; /* TALK ONLY  */
36  1           OUTPUT  (COMMAND$MOD)        =PRESCALER;
37  1           OUTPUT  (COMMAND$MOD)        =HIGH$SPEED;
38  1           OUTPUT  (COMMAND$MOD)        =PON;

39  1           DO WHILE  (INPUT  (STATUS$1)  AND BO)   =0;
40  2           END;    /* WAIT FOR BO INTR  */
41  1           OUTPUT  (PORT$OUT)   =0AAH;

42  1           DO WHILE  (INPUT  (STATUS$1)  AND ERR)   =ERR;
43  2             DO WHILE   (INPUT  (STATUS$1)   AND BO)   =0;
44  3             END;    /* WAIT FOR BO INTR  */
45  2             OUTPUT  (PORT$OUT)  =0AAH;
46  2           END;

47  1           OUTPUT  (STATUS$2)    =DMA$REG$T;
                /*  WAIT FOR TC=0  */

48  1               DO WHILE   (INPUT   (CMD$37)  AND TC) <>  TC;
49  2               END;

50  1           GOTO START91;

51  1    END;


MODULE INFORMATION

    CODE AREA SIZE       =0122H    290D
    CONSTANT AREA SIZE   =0000H      0D
    VARIABLE AREA SIZE   =0001H      1D
    MAXIMUM STACK SIZE   =0000H      0D
    152 LINES READ
    0 PROGRAM ERROR (S)
```

APPENDIX 3
SOFTWARE FOR HP 9835A

```
10   REM SEND IN
TERFACE CLEAR
20   ABORTIO 7
30   REM FORCE E
RRORS UNTIL LIST
ENERS ACTIVE
40 Frcerr:   OUT
PUT 704 USING "#
,K";"B"
50 Chkstst:   ST
ATUS 7;Stat1,Sta
t2,Stat3,Stat4
60   Err=Stat2 A
ND 1
70   IF Err=1 TH
EN GOTO Frcerr
80   PRINT CHR$(
12),"LISTENERS A
RE ON LINE "
90   REM CONFIGU
RE PPOLL
100  PPOLL CONFI
GURE 704;"000001
00"
110

   ! response on
 bit 4
120  PRINT CHR$(
12),"PARALLEL PO
LL CONFIGURED"
130  REM ENABLE
KEYBOARD INTERRU
PT
140  PRINT "COMM
AND = ?      (HIT
 'H' FOR LIST)"
150 Keyen:   ON K
BD GOSUB 610
160  STATUS 7;St
at1,Stat2,Stat3,
Stat4
170  Srq=BINAND(
Stat1,128)
180  IF Srq=0 TH
EN GOTO Keyen
190  OFF KBD
---
200  PRINT CHR$(
12),"SRQ RECEIVE
D"
210  PRINT "SEND
ING PARALLEL POL
L RESPONSE MESSA
GE"
220  REM EXECUTI
NG PARALLEL POLL
230  Ppollbyte=P
POLL(7)
240  PRINT "PARA
LLEL POLL BYTE =
 ";Ppollbyte
250  PRINT "____
----------------
------"
260  Ppollbyte=B
INAND(Ppollbyte,
8)
270  IF Ppollbyt
e=0 THEN GOTO PS
291
280  PRINT "SRQ
NOT FROM 8291"
281  PRINT "COMM
AND = ?      (HIT
 'H' FOR LIST)"
290  GOTO Keyen
300 PS291:   PRIN
T "SRQ IS FROM N
CC 8291 ... THE
ENTERPRISE"
310  PRINT "PERF
```

```
ORMING SERIAL PO
LL TO GET STATUS
 "
320  STATUS 704;
Stat
330  PRINT CHR$(
12),"Status = ";
Stat
340  Dxfer=BINAN
D(Stat,1)
520  IF Dxfer>0
THEN GOTO Rcvr
530  GOTO Keyen
531 Rcvr:  REM R
EADY TO RCV CHAR
S FROM GPIB
540  DIM G$[80]
550  ENTER 704 U
SING "%,T";G$
560  PRINT CHR$(
12),G$
570  PRINT "COMM
AND = ?     (HIT
 'H' FOR LIST)"
580  GOTO Keyen
590  REM INTERRU
PT SERVICE ROUTI
NES
600  REM GET KEY
BOARD DATA
610 Whatkey:  DI
M K$[80]
620  K$=KBD$
630  IF K$="G" T
HEN GOTO Get
640  IF K$="D" T
HEN GOTO Dec
650  IF K$="R" T
HEN GOTO Rem
660  IF K$="H" T
HEN GOTO Help
670  IF K$="X" T
HEN GOTO Xmit
680 Get:  TRIGGE
R 704
690  PRINT CHR$(
12),"GROUP EXECU
TE TRIGGER SENT"
700  PRINT " "
710  PRINT "COMM
AND = ?     (HIT
'H' FOR LIST)"
720  RETURN
730 Dec:  RESET
704
740  PRINT CHR$(
12),"SELECTIVE D
EVICE CLEAR SENT
"
750  PRINT " "
760  PRINT "COMM
AND = ?     (HIT
'H' FOR LIST)"
770  RETURN
780 Rem:  LOCAL
704
790  PRINT CHR$(
12),"REMOTE MESS
AGE SENT"
800  PRINT " "
810  PRINT "COMM
AND = ?     (HIT
 'H' FOR LIST)"
820  RETURN
830 Help:  PRINT
 CHR$(12)
840  PRINT " @@@
@ OPERATOR ALLOW
ABLE COMMANDS @@
@@ "
850  PRINT " hit
 key   result"
860  PRINT "   G
      Send GET m
essage"
870  PRINT "   D
      Send DEC m
essage"
```

```
880  PRINT "    R
  `    Send REM/L
OC message"
890  PRINT "    X
       Xmits keyb
oard input to 82
91"
900  PRINT "    H
       Prints thi
s table"
910  PRINT " "
920  PRINT "...
go ahead, TRY IT
 !"
930  RETURN
940 Xmit:  DIM A
$[80]
950  PRINT CHR$(
12),"Enter data
to send and hit
CONTINUE"
960  INPUT A$
970  OUTPUT 704;
A$
971  EOI 7;0
980  PRINT "COMM
AND = ?      (HIT
 'H' FOR LIST)"
990  RETURN
1000 END
```

APPENDIX 4
SOFTWARE FOR HP 8088/HP 9835A VIA GPIB

```
PL/M-86 COMPILER     HPIB


ISIS-II PL/M-86 V1.1 COMPILATION OF MODULE HPIB
OBJECT MODULE PLACED IN :F1:HPIB.OBJ
COMPILER INVOKED BY:  PLM86 :F1:HPIB.SRC LARGE


   1            HPIB:
                /*

                PARAMETER DECLARATIONS
                */

                DO;

   2    1       DECLARE

                ADDR$HI       LITERALLY    '01H',
                ADDR$LO       LITERALLY    '00H',
                ADSC          LITERALLY    '01H',
                BI        LITERALLY    '01H',
                BO        LITERALLY    '02H',
                CHAR$COUNT    BYTE,
                CHAR          BYTE,
                CHARS(80)     BYTE,
                CLEAR         LITERALLY    '00H',
                CPT       LITERALLY    '80H',
                CRLF          LITERALLY    '0AH',
                DEC       LITERALLY    '08H',
                DMA$ADR$LSTN      POINTER,
                DMA$ADR$TALK      POINTER,
                DMA$WRD$LSTN(2) WORD AT       (@DMA$ADR$LSTN),
                DMA$WRD$TALK(2) WORD AT       (@DMA$ADR$TALK),
                DMA$REQ$L     LITERALLY    '10H',
                DMA$REQ$T     LITERALLY    '20H',
                DNE       LITERALLY    '10H',
                END$EOI       LITERALLY    '88H',
                EOS       LITERALLY    '0DH',
                ERR       LITERALLY    '04H',
                GET       LITERALLY    '20H',
                I         BYTE,
                LISTEN        LITERALLY    '04H',
                MLA       LITERALLY    '04H',
                MODE$1        LITERALLY    '01H',
                NO$DMA        LITERALLY    '00H',
                NO$RSV        LITERALLY    '00H',
                NORM$TIME     LITERALLY    '20H',
                PON       LITERALLY    '00H',
                PPC       LITERALLY    '05H',
                PPE$MASK      LITERALLY    '60H',
                PPOLL$CNFG$FLAG LITERALLY    '01H',
                PPOLL$EN$BYTE     BYTE,
                PRI$BUF(80) BYTE AT       (@CHARS),
                RD$XFER       LITERALLY    '48H',
                RESET         LITERALLY    '02H',
                REMC          LITERALLY    '02H',
                RSV       LITERALLY    '40H',
                RXRDY         LITERALLY    '02H',
```

```
PL/M-86 COMPILER    HPIB

                SRQS            LITERALLY   '40H',
                STAT1           BYTE,
                STAT2           BYTE,
                TALK            LITERALLY   '02H',
                TA$OR$LA        BYTE,
                TRQ         LITERALLY   '41H',
                TC          LITERALLY   '01H',
                TC$HI           LITERALLY   '00H',
                TC$LO           LITERALLY   '0FFH',
                TXRDY           LITERALLY   '01H',
                UDC         BYTE,
                WR$XFER         LITERALLY   '44H',
                XYZ         BYTE;

                /*

                PORT DECLARATIONS

                */

  3   1         DECLARE

                ADDR$O          LITERALLY   '0FFC6H',
                ADDR$STATUS     LITERALLY   '0FFC4H',
                CLEAR$FF        LITERALLY   '0FFDDH',
                CMD$37          LITERALLY   '0FFD8H',
                COMMAND$MOD     LITERALLY   '0FFC5H',
                COUNT$HI        LITERALLY   '0FFD1H',
                COUNT$LO        LITERALLY   '0FFD1H',
                CPT$REG         LITERALLY   '0FFC5H',
                EOS$REG         LITERALLY   '0FFC7H',
                PORT$IN         LITERALLY   '0FFC0H',
                PORT$OUT        LITERALLY   '0FFC0H',
                SER$DATA        LITERALLY   '0FFF0H',
                SER$STAT        LITERALLY   '0FFF2H',
                SET$MASK        LITERALLY   '0FFDFH',
                SET$MODE        LITERALLY   '0FFDBH',
                SPOLL$STAT      LITERALLY   '0FFC3H',
                START$HI        LITERALLY   '0FFD0H',
                START$LO        LITERALLY   '0FFD0H',
                STATUS$1        LITERALLY   '0FFC1H',
                STATUS$2        LITERALLY   '0FFC2H';

                /* crt  messages list */

  4   1         DECLARE GET$MSG(11) BYTE DATA (0DH,0AH,'TRIGGER',0AH,0DH);
  5   1         DECLARE DEC$MSG(16) BYTE DATA (0DH,0AH,'DEVICE CLEAR',0AH,0DH);
  6   1         DECLARE REMC$MSG(10) BYTE DATA (0DH,0AH,'REMOTE',0DH,0AH);
  7   1         DECLARE CPT$MSG(22) BYTE DATA (0DH,0AH,'UNDEF CMD RECEIVED',0AH,0DH)
  8   1         DECLARE HUH$MSG(11) BYTE DATA (0DH,0AH,'HUH ???',0DH,0AH);

                    /* called procedures */

  9   1         REQSER:     PROCEDURE;
```

```
PL/M-86 COMPILER     HPIB

  10    2               OUTPUT (SPOLL$STAT)=TRQ;

  11    2               DO WHILE (INPUT (SPOLL$STAT) AND SRQS)=SRQS;
  12    3               END;

  13    2               OUTPUT (SPOLL$STAT)=NO$RSV;

  14    2               END REQSER;

  15    1       CO: PROCEDURE(XXX);
  16    2               DECLARE
                         XXX        BYTE;

  17    2               DO WHILE (INPUT (SER$STAT) AND TXRDY)<>TXRDY;
  18    3               END;
  19    2               OUTPUT (SER$DATA)=XXX;
  20    2           END CO;
  21    1       HUH:        PROCEDURE;
  22    2                   DO I=0 TO 10;
  23    3                    CALL CO (HUH$MSG(I));
  24    3                   END;
  25    2               END HUH;

  26    1       CI:     PROCEDURE;

  27    2               IF (INPUT (SER$STAT) AND RXRDY)=RXRDY THEN
  28    2                   DO;
  29    3                       I=0;
  30    3                       CHAR$COUNT=0;
  31    3       STORE$CHAR:         CHAR=(INPUT (SER$DATA) AND 7FH);
  32    3                       CHAR$COUNT=CHAR$COUNT+1;
  33    3                       CALL CO (CHAR);
  34    3                       CHARS(I)=CHAR;
  35    3                       I=I+1;
  36    3                       IF CHAR <> CRLF THEN
  37    3                           DO;
  38    4                            DO WHILE (INPUT (SER$STAT) AND RXRDY) <>RXRDY;
  39    5                            END;
  40    4                            GOTO STORE$CHAR;
  41    4                           END;
  42    3                       CALL REQSER;
  43    3                   END;
  44    2               END CI;

  45    1       TALK$EXEC:  PROCEDURE;

  46    2               OUTPUT (STATUS$2)=CLEAR;

                        /*
                        manipulate address bits for DMA controller
                        */

  47    2               DMA$ADR$TALK=(@CHARS);
  48    2               DMA$WRD$TALK(1)=SHL(DMA$WRD$TALK(1),4);
  49    2               DMA$WRD$TALK(0)=DMA$WRD$TALK(0)+DMA$WRD$TALK(1);

  50    2               OUTPUT (CLEAR$FF)=CLEAR;
```

```
PL/M-86 COMPILER     HPIB

  51   2              OUTPUT (CMD37)=NORM$TIME;
  52   2              OUTPUT (SET$MODE)=RD$XFER;
  53   2              OUTPUT (SET$MASK)=CLEAR;
  54   2              OUTPUT (START$LO)=DMA$WRD$TALK(0);
  55   2              DMA$WRD$TALK(0)=SHR(DMA$WRD$TALK(0),8);
  56   2              OUTPUT (START$HI)=DMA$WRD$TALK(0);
  57   2              OUTPUT (COUNT$LO)=CHAR$COUNT;
  58   2              OUTPUT (COUNT$HI)=0;


  59   2              OUTPUT (EOS$REG)=EOS;
  60   2              OUTPUT (COMMAND$MOD)=END$EOI;

  61   2              DO WHILE (INPUT (STATUS$1) AND BO)=0;
  62   3              END;
  63   2              OUTPUT (PORT$OUT)=0AAH;

  64   2              DO WHILE (INPUT (STATUS$1) AND ERR)=ERR;
  65   3                  DO WHILE (INPUT (STATUS$1) AND BO)=0;
  66   4                  END;
  67   3                  OUTPUT (PORT$OUT)=0AAH;
  68   3              END;
  69   2              OUTPUT (STATUS$2)=DMA$REQ$T;

  70   2              END TALK$EXEC;

  71   1      LISTEN$EXEC:     PROCEDURE;

  72   2              OUTPUT (STATUS$2)=CLEAR;
  73   2              OUTPUT (CLEAR$FF)=CLEAR;
  74   2              OUTPUT (CMD$37)=NORM$TIME;
  75   2              OUTPUT (SET$MODE)=WR$XFER;
  76   2              OUTPUT (SET$MASK)=CLEAR;
  77   2              DMA$ADR$LSTN=(@CHARS);
  78   2              DMA$WRD$LSTN(1)=SHL(DMA$WRD$LSTN(1),4);
  79   2              DMA$WRD$LSTN(0)=DMA$WRD$LSTN(0)+DMA$WRD$LSTN(1);
  80   2              OUTPUT (START$LO)=DMA$WRD$LSTN(0);
  81   2              DMA$WRD$LSTN(0)=SHR(DMA$WRD$LSTN(0),8);
  82   2              OUTPUT (START$HI)=DMA$WRD$LSTN(0);
  83   2              OUTPUT (COUNT$LO)=TC$LO;
  84   2              OUTPUT (COUNT$HI)=TC$HI;
  85   2              OUTPUT (STATUS$2)=DMA$REQ$L;

  86   2              END LISTEN$EXEC;

  87   1      PRINTER:     PROCEDURE;

  88   2              I=0;

  89   2              DO WHILE PRI$BUF(I) <>CRLF;
  90   3                  CALL CO (PRI$BUF(I));
  91   3                  I=I+1;
  92   3              END;
  93   2              CALL CO (PRI$BUF(I));

  94   2              END PRINTER;
```

```
PL/M-86 COMPILER    HPIB


 95   1      ADSC$EXEC:  PROCEDURE;

 96   2              TA$OR$LA=INPUT (ADDR$STATUS);

 97   2              IF (TA$OR$LA AND TALK)=TALK THEN
 98   2                CALL TALK$EXEC;
 99   2              IF (TA$OR$LA AND LISTEN)=LISTEN THEN
100   2                CALL LISTEN$EXEC;

101   2              END ADSC$EXEC;

102   1      GET$EXEC:   PROCEDURE;
103   2                  DO I=0 TO 10;
104   3                   CALL CO (GET$MSG(I));
105   3                  END;
106   2            END GET$EXEC;

107   1      DEC$EXEC:   PROCEDURE;
108   2                  DO I=0 TO 15;
109   3                   CALL CO (DEC$MSG(I));
110   3                  END;
111   2            END DEC$EXEC;

112   1      REMC$EXEC:  PROCEDURE;
113   2                  DO I=0 TO 9;
114   3                   CALL CO (REMC$MSG(I));
115   3                  END;
116   2            END REMC$EXEC;

117   1      PPOLL$CON:  PROCEDURE;

118   2              OUTPUT (COMMAND$MOD)=PPOLL$CNFG$FLAG;

119   2              END PPOLL$CON;

120   1      PPOLL$EN:   PROCEDURE;

121   2              PPOLL$EN$BYTE=(UDC AND 6FH);
122   2              OUTPUT (COMMAND$MOD)=PPOLL$EN$BYTE;

123   2              END PPOLL$EN;

124   1      CPT$EXEC:   PROCEDURE;
125   2                  DO I=0 TO 21;
126   3                   CALL CO (CPT$MSG(I));
127   3                  END;

128   2                  UDC=INPUT (CPT$REG);
129   2                  UDC=(UDC AND 7FH);
130   2                  IF (UDC AND PPC)=PPC THEN
131   2                   CALL PPOLL$CON;

132   2                  IF (UDC AND PPE$MASK)=PPE$MASK THEN
133   2                   CALL PPOLL$EN;
```

```
PL/M-86 COMPILER     HPIB


 134   2                END CPT$EXEC;
                  /*
                  BEGIN CODE
                  */

 135   1          INIT:

                      OUTPUT   (CLEAR$FF)   =CLEAR;
 136   1                OUTPUT   (COMMAND$MOD)    =RESET;
 137   1                OUTPUT   (ADDR$STATUS)    =MODE$1;
 138   1                OUTPUT   (ADDR$O)     =MLA;
 139   1                OUTPUT   (STATUS$2)   =NO$DMA;
 140   1                OUTPUT   (COMMAND$MOD)    =PON;

 141   1          LISTENERS:

                      /* response to listeners check */


                      DO WHILE (INPUT (STATUS$1) AND BI)=0;
 142   2                END;

 143   1                XYZ=INPUT (PORT$IN);
 144   1                XYZ=INPUT (STATUS$2);

 145   1          CMD:

                  RDSTAT:
                      /* read status registers and interpret command */

                      STAT1=INPUT (STATUS$1);
 146   1                STAT2=INPUT (STATUS$2);

 147   1                IF (STAT1 AND DNE)=DNE THEN
 148   1                   CALL PRINTER;
 149   1                IF (STAT1 AND CPT)=CPT THEN
 150   1                     DO;
 151   2                     CALL CPT$EXEC;
 152   2                     STAT2=(STAT2 AND OFEH);
 153   2                     END;
 154   1                IF (STAT1 AND GET)=GET THEN
 155   1                     DO;
 156   2                     CALL GET$EXEC;
 157   2                     STAT2=(STAT2 AND OFEH);
 158   2                     END;
 159   1                IF (STAT1 AND DEC)=DEC THEN
 160   1                     DO;
 161   2                     CALL DEC$EXEC;
 162   2                     STAT2=(STAT2 AND OFEH);
 163   2                     END;
 164   1                IF (STAT2 AND REMC)=REMC THEN
 165   1                     DO;
 166   2                     CALL REMC$EXEC;
 167   2                     STAT2=(STAT2 AND OFEH);
 168   2                     END;
 169   1                IF (STAT2 AND ADSC)=ADSC THEN
```

```
PL/M-86 COMPILER    HPIB

 170   1              DO;
 171   2              CALL ADSC$EXEC;
 172   2              STAT2=(STAT2 AND OFEH);
 173   2              END;

 174   1          CALL CI;

 175   1          GOTO CMD;



 176   1      END;


MODULE INFORMATION:

     CODE AREA SIZE     = 0475H   1141D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 0061H     97D
     MAXIMUM STACK SIZE = 000AH     10D
     349 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-86 COMPILATION
```